**Abstract**

An Experimental Application of Machine Learning-Based Classification Techniques to Proactively Identify Insider Threats

Joshua D. Bowen

Northern Arizona University

Prescott Valley Campus

Insider incidents are on the rise. Just like the high-profile security breaches such as Snowden, thousands of insider-perpetrated security breaches occur in United States businesses every day. While current commercial software can monitor, log, and prevent access to designated files and directories, it remains difficult to predict and prevent unauthorized insider usage. Due to the gaps in research in this area, the focus of this study is to more accurately predict insider threats within a terminal environment.

Linux was chosen specifically because of its ubiquity on commercial servers around the globe. Amazon’s Machine Learning (AML) service has been selected to analyze the data, reduce the necessary computing power, and to minimize human factors considerations in the design of the machine learning (ML) architecture. AML uses multinomial logistic regression for multi-class classification and uses the stochastic gradient descent optimization technique.

If it can be demonstrated that ML software can accurately identify and predict insider threats, this research could be a foundation for future cyber security software architectures. Other opportunities for research in this area would include ML applications in intruder and malware detection.

**Introduction**

Much of the research into insider threats fall into several distinct categories. Analysis of the technological systems, physiological analysis of the user, and methods combining both such as Brdiczka et. al. [1]. None of the approaches have been tested using a practical prototype nor attempt proactive detection. This is why this project attempts to fill both those gaps.

In order to detect an insider threat one must define what such a threat is. The definition used by Probst et. al. seems to fit well. “an insider threat is [posed by] an individual with privileges who misuses them or whose

access results in misuse” [5].

This project applies a proactive approach to intercept and evaluate a user’s command before execution. This allows mistakes and malicious activities to be stopped before causing damage.

This project takes the approach of detecting insider threats based on commands similarly to Schonlau et. al. [13] and Maxion, Townsend [14]. It also applies machine learning similar to Parveen et. al. [3].

**Data** **Gathering**

This project utilizes three main sources of data.

* The first is a file that contains an un-edited history of used commands, taken from the computer the prototype is tested on.
* Second is a list of mistake and malicious commands assembled during this research because of the lack of such examples in the other datasets.
* The third is the UNIX User Dataset from Purdue University. It is comprised of commands captured over a two year period from several different users at Purdue.

These datasets will be converted to a CSV (comma separated values) format for use by the ML model. Simple scripts were developed to remove unnecessary lines and markup from the Purdue and the test computer’s datasets. The same scripts used a rules-based system to label all the commands in each dataset. In all there are about 150,000 data points.

**Methods**

The method utilizes the AML software. It will be trained on the datasets which are comprised of normal user situations, crafted mistakes, and simulated malicious activity. A testing dataset of 100 commands will be given to the model for batch prediction and will be evaluated for statistical accuracy.

A prototype was developed to use the AML software to make predictions against given commands to verify accuracy. This prototype will use data available from the operating system to mimic the usual command prompt in a terminal. The prototype will capture user input, send the command to the AML software which will return a set of probabilities for each possible category. Due to the low volume of data in the mistake and malicious categories, special weighting will be applied to the probabilities to make a final determination. Based on that the prototype will either allow the command to execute, warn the user of a mistake, or cutoff access in the case of a suspected malicious command.

The prototype will be tested against a human actor to test predictions in a high-fidelity simulation. The python timeit module will be used to test for latency between the user entering their command and when it executed. Four categories will be analyzed; code only, execution only, code + execution, and code + execution + AML. To have a baseline for the network speed, the ping command will be used to analyze the round trip time for a signal to travel between the test site and the AML server.

**Results**

The batch predictions show that the model is 100% accurate at predicting commands in the normal use category. The model has a 30% rate of false negatives for both malicious and mistake commands. It did not predict false positives in any category. This is due to the lack of data in the mistake and malicious categories, the model is unable to predict with more than ~35% certainty that any one command is in those two categories.

The results of the high-fidelity simulation indicate that the prototype is partially effective at capturing all input. The participants were able to escape it and, after that, became untraceable to the prototype. Future versions will address this issue.

In the latency tests, results show a small discrepancy of an additional ~0.25ms between the combined times of code only and execution only, compared with the code + execution time.

There is a major difference between the network speed, execution, and the time it takes to interact with the AML server of over ~440ms.

The process has indicated that a prototype like this can capture all commands however it cannot execute them all properly due to the method of capture. This problem affects about 20% of commands.

**Conclusions**

The results of the test indicate the model isn't sufficiently accurate in detecting malicious and mistake activity even when given special weighting. This is likely due to the lack of data in those categories.

Future research in the area needs more open data. One solution is to ask organizations to publish command logs after an attack. Another solution would be to apply transfer learning, taking a model trained on one type of data an applying it to another [11]. This has the potential to increase the quality of predictions without needing more data.

In addition to the lack of data another problem is that more than two thirds of the data used in this project was cleaned, leaving it with little to no context. Unmodified command histories like the one taken from the computer the tests were run on would help increase the quality of predictions from the ML model.

There are a few factors that come into account with the latency of the prototype. One is that most of the code in the evaluate() function was commented out and the get\_prediction() and get\_input() functions were not called at all when testing code execution times but were all used when AML was added to the testing. While the high-latency is understandable, this is unacceptable and would be highly noticeable to all users. In the future, using a white-list of known good commands that don’t need evaluation would reduce time. One possible solution would be to rewrite the software in a low-level language like C or Rust to take advantage of memory and other optimizations. Another option would be to allow command execution before getting results from AML, eliminating the latency issue altogether.

This study also indicated that further improvements are needed to the prototype to accurately capture information from a live terminal. While the prototype created a fake prompt, it was severely limited in what it could capture because not all commands could be run correctly. This problem skewed the data from the simulation. Further work on this prototype could decrease latency, allow all commands to be executed properly, and prevent the user from escaping command capture.

Due to the poor latency the prototype is impractical on the user experience aspect. Also because it is reliant on a network connection to the AML service it would not be practical for use on laptops or mobile devices. It could also be bypassed on desktop computers by cutting off network access. However if used exclusively on servers that are permanently connected to the internet, the reliance issue would not exist.

Another shortfall of this approach is that commands are evaluated individually, because of that the model was not trained to look at larger patterns. Further work in looking at command history could improve the quality of predictions. It would also be necessary if this was used in a commercial environment because each user is going to have a different baseline of normal behavior.

Future research is needed in order to determine how to best approach the problem of practical proactive

insider threat detection. As other researchers have indicated [4], larger datasets comprising of more points such as time-stamps and keyboard information would help better track user behavior in an attempt to establish recognizable patterns by ML models. This and other research indicate that more data is needed.

**Terms**

*Terminal* – a widely used virtual console that mimics the text based user interfaces of the 1980s.

*Linux* – this term is the official name for the UNIX-like computing kernel developed by Linus Torvalds. It is widely used to reference any operating system based on that kernel.

*Python* – a popular high-level programming language.

*Functions* – a logical construct used to complete a specific task in computer code.

*Multinomial logistic regression –* a statistical classification method that applies logistic regression to multi-class problems (meaning problems with more than one outcome).

*Stochastic gradient descent optimization* – an algorithm that tries to find minima or maxima by iteration.

**Funding**

The work reported in this study was initiated and self-funded by the author.

**Acknowledgements**

Faculty Advisor - James A. Subach, Ph. D

Academic Credit Advisor - Nancy L. Jensen

Code Help - Ian Harvey on Stack Overflow

Data Sources - Purdue University

Poster Template - Felix Breuer

The NAU Cyber Security Team

The Embry-Riddle [Ethical] Hackers Club

Amazon Web Services

Font Awesome

GitHub

**Source Code & Materials are available at:** <http://github.com/Skraelingjar/cyberml>

**References**

1. Brdiczka O, Juan Liu, Price B, et al. Proactive insider threat detection through graph learning and psychological context. spw. 2012:142-149. http://ieeexplore.ieee.org.libproxy.nau.edu/document/6227698. doi: 10.1109/SPW.2012.29.

2. Rashid T, Agrafiotis I, Nurse J. A new take on detecting insider threats. Proceedings of the 8th ACM CCS International Workshop on managing insider security threats. Oct 28, 2016:47-56. http://dl.acm.org.libproxy.nau.edu/citation.cfm?id=2995964. doi: 10.1145/2995959.2995964.

3. Parveen P, Weger ZR, Thuraisingham B, Hamlen K, Khan L. Supervised learning for insider threat detection using stream mining. ICTAI. 2011:1032-1039. http://ieeexplore.ieee.org.libproxy.nau.edu/document/6103467. doi: 10.1109/ICTAI.2011.176.

4. Eldardiry H, Bart E, Juan Liu, Hanley J, Price B, Brdiczka O. Multi-domain information fusion for insider threat detection. spw. 2013:45-51. http://ieeexplore.ieee.org.libproxy.nau.edu/document/6565228. doi: 10.1109/SPW.2013.14.

5. Probst CW, Bishop M, Gollmann D, Hunker JA. Insider threats in cyber security. Vol 49. Boston: Springer Science+Business Media; 2010. http://lib.myilibrary.com.libproxy.nau.edu?ID=790801.

6. Maybury M, Chase P, Cheikes B, et al. Analysis and detection of malicious insiders. . 2005. http://www.dtic.mil/docs/citations/ADA456356.

7. Greengard S. Cybersecurity gets smart. Communications of the ACM. 2016;59(5):29-31. http://dl.acm.org.libproxy.nau.edu/citation.cfm?id=2898969. doi: 10.1145/2898969.

8. Rash W. 'Deep learning' technology sees through security software blind spots. eWeek. 2015:1.

9. Preimesberger C. Eight tactics for circumventing insider security threats. eWeek. 2015:1.

10. Hackers offering money to company insiders in return for confidential data. The Hacker News.

11. Sinno Jialin Pan, Qiang Yang. A survey on transfer learning. TKDE. 2010;22(10):1345-1359. http://ieeexplore.ieee.org.libproxy.nau.edu/document/5288526. doi: 10.1109/TKDE.2009.191.

12. Rick Dana Barlow. Grand theft data. Health Management Technology. 2014;35(2):8. http://search.proquest.com.libproxy.nau.edu/docview/1542396962.

13. M. Schonlau, W. DuMouchel, et. al. Computer Intrusion: Detecting masquerades. Statistical Science 2001;16(1):58-74.

14. Maxion RA, Townsend TN. Masquerade detection augmented with error analysis. TR. 2004;53(1):124-147. http://ieeexplore.ieee.org/document/1282170. doi: 10.1109/TR.2004.824828.